编译原理实验三报告

191870271 赵超懿

实现功能

1. 完成了中间代码的生成
2. 除了基本的中间代码生成要求，还完成了要求3.1，3.2，实现了对数组和结构体的操作
3. 对中间代码进行了一定的优化，如控制流穿越

程序使用及编译

1. 在Code文件夹下运行make parser生成二进制文件parser
2. 运行命令./paser $FILE $FILE即可
3. make test做了一些修改，详细情况见makefile

程序特点及具体实现

程序特点

1. 在生成代码的过程中，实现了控制流穿越的优化，参考了龙书上的方案，经过测试，可以有效减少指令数量
2. 优化：一种简单的思路是因为在中间代码生成的过程中，一定会产生冗余的指令，如将常数赋值给一个临时变量。另一种是使用龙书介绍的方式，即DAG来进行局部子表达式的消除
3. 实现了对结构体的赋值和连续赋值，如a = b= c

具体实现

我的代码采用了线性IR的方式生成，和语义分析中一样，使用类似于SDD的方式，每个节点对应一定的动作。

赋值的实现

赋值的主要问题是在于等号左右都有可能出现结构体和数组，而在类型为Exp的节点，定义的函数为translate_Exp(Node *,int * place). 由于place的存在，需要将等号的左右进行区分，对于等号左边的，如果是结构体的成员或数组的某个元素，那么需要获得其地址，再对地址对应的值赋值，如果是普通变量，只需要对变量赋值。这里我采用了类似于SDD的方案，使用继承属性的思想，通过在当前节点上层节点给出当前节点的类型（即需要地址还是变量），来控制返回的是变量还是地址。同时使用综合属性的思想，来得到子节点的变量的类型，来判断是否需要进行解引用的操作。

```
if(type(left->lchild,"ID") && left->syn->kind == BASIC) {

gencode(T_ASSIGN,genoperand(VARIABLE,t1),genoperand(INT_CONST,num))
;
} else if(left->syn->kind == BASIC) {
    gencode(T_STAR_A,genoperand(VARIABLE,t1),
genoperand(INT_CONST,num));
}
```

函数调用中的问题

普通的函数调用不会产生问题，但是对于类似于这样的调用

```
a = func1(func2(),func3())
```

参数需要调整好顺序，这里的实现我选择了全局变量的栈，会先计算所有的参数，只有在正式调用函数时，会根据参数的数量，将栈中对应的变量pop出来

代码的生成

```
Operand genoperand(int kind,...);  //生成操作数
static void gencode(int kind,...);  //单条代码的生成
InterCode * u_gencode(int kind,va_list ap);  //返回生成的代码
```

这里实现全部使用可变变量以减少可重复的代码量

代码的优化

控制流穿越

这个在书上已经有很详细的介绍，实现的过程只需要增加一个reverse RELOP符号 的函数，然后再修改IF,WHILE,AND,OR等的代码生即可。

DAG

这个我做的不好，首先是实现不了SSA，然后是目前还有一些小bug。

具体实现是先进行块的划分，然后对每个块进行DAG的构建（由于缺少map，vector这样的数据结构，写的很难受），将代码的类型分为四类，第一类是加减乘除，第二类是地址赋值，第三类是单独的赋值，最后是其他的代码，对不同的代码进行不同的操作（手写的hashmap和vector很难用）

一点建议

用C写实验有些难受，缺少很多工具，希望可以使用C++来完成实验（GCC也用的C++）